# Using MongoDB Sync to Replace Kinesis for Real-Time Fuel Data Update

**Rohith Varma Vegesna**

(Software Engineer 2)
Texas, USA
rohithvegesna@gmail.com

**Abstract**

**The need for real-time data synchronization in fuel stations has led to widespread adoption of cloud-based streaming services such as AWS Kinesis, which allows for high-throughput data ingestion and event processing but at the cost of increased complexity, latency, and additional operational expenses. Fuel controllers generate a constant stream of transactional data that must be transmitted to the cloud efficiently and with minimal delay. However, the reliance on Kinesis and Lambda introduces multiple processing stages that can degrade real-time performance. This paper explores an alternative approach using MongoDB Sync, enabling direct synchronization from local MongoDB instances within fuel controllers to cloud-hosted MongoDB databases. By eliminating the need for event-driven computation layers, such as Lambda, and reducing dependency on cloud-managed streaming solutions, MongoDB Sync offers a simplified, cost-effective, and low-latency data update mechanism. The proposed architecture ensures that fuel transactions and ATG data are continuously and reliably synchronized without the need for intermediary processing layers. This study examines the challenges of implementing MongoDB Sync in fuel station environments, evaluates key performance metrics such as latency and resource utilization, and presents a comparative analysis with Kinesis-based solutions to determine the effectiveness of the proposed approach in enhancing operational efficiency while minimizing costs.**

**Keywords: MongoDB Sync, Fuel Controller, Cloud Synchronization, Real-Time Data Update, Fuel Station Monitoring**

## 1. Introduction

### 1.1 Background

Fuel station management systems rely on Automated Tank Gauges (ATGs) and pump dispensers to track fuel levels, sales, and operational metrics, which are essential for ensuring efficient operations and preventing fuel loss. These systems traditionally push data to cloud environments using streaming architectures such as AWS Kinesis, which provides high-throughput ingestion and facilitates real-time data processing. However, while Kinesis allows for scalable streaming, its reliance on AWS Lambda for event processing adds significant complexity, introduces additional latency, and increases operational costs, especially for fuel stations handling large volumes of transactions and fuel level updates.

The inherent limitations of event-driven architectures necessitate an alternative that simplifies infrastructure and improves efficiency. A more direct approach using MongoDB Sync can enable real-time updates by synchronizing local MongoDB instances within fuel controllers to cloud-hosted MongoDB databases without requiring an intermediary streaming service. This approach removes the need for event-driven data handling and eliminates processing delays introduced by multi-step ingestion and transformation mechanisms, resulting in significantly improved real-time responsiveness for fuel management systems.

By leveraging MongoDB Sync, fuel stations can maintain a seamless data flow from local controllers to the cloud while reducing overall system complexity. The real-time data synchronization ensures that fuel transactions, ATG readings, and dispenser activities are promptly reflected in the cloud database, allowing for improved analytics, monitoring, and operational decision-making. Furthermore, reducing reliance on third-party streaming solutions not only enhances system efficiency but also lowers long-term infrastructure costs, making MongoDB Sync a compelling alternative to traditional streaming-based architectures for fuel station management.

### 1.2 Problem Statement

AWS Kinesis and Lambda introduce non-trivial operational overheads when handling real-time fuel transaction data, as they involve multiple steps of ingestion, transformation, and processing before the data is made available in the cloud. Fuel controllers generate high-frequency transactional data that requires immediate synchronization with cloud databases to maintain operational efficiency and support real-time decision-making. However, the existing pipeline using Kinesis requires data ingestion, event-driven processing with AWS Lambda, and storage into a cloud database, all of which contribute to increased response times and potential processing bottlenecks. Additionally, reliance on event-driven architectures results in increased complexity, higher infrastructure costs, and potential delays due to cold starts in Lambda executions. This paper investigates how MongoDB Sync can replace Kinesis and Lambda by enabling direct synchronization between local MongoDB instances within fuel controllers and cloud MongoDB clusters, thereby reducing latency, simplifying data flows, and minimizing the need for additional cloud services. The study further evaluates the efficiency gains achieved through this transition, including improvements in system responsiveness, reductions in cloud dependency, and enhanced real-time monitoring capabilities for fuel station operations.

### 1.3 Objectives

- To design a MongoDB-based synchronization mechanism for real-time fuel data updates.
- To compare performance metrics between MongoDB Sync and AWS Kinesis-based architectures.
- To evaluate latency improvements and operational efficiency.
- To implement a case study demonstrating the transition from a Kinesis-based to a MongoDB Sync-based architecture.

## 2. Literature Review

Several real-time data streaming solutions have been proposed for fuel management systems, each offering unique advantages and posing distinct challenges. Traditional cloud-based streaming services have been widely adopted due to their ability to handle large-scale data ingestion efficiently. These systems provide structured and event-driven pipelines that process and store incoming data, ensuring availability for downstream applications. However, such architectures often come with significant overhead, including increased computational load, elevated operational costs, and potential performance bottlenecks due to multi-step processing.

These multi-stage workflows involve initial ingestion, transformation, and storage in separate layers, requiring additional infrastructure to ensure proper synchronization between components. This added complexity not only increases system maintenance challenges but also extends processing times, causing delays in real-time fuel data updates. Studies have examined the impact of event-driven processing, highlighting inefficiencies associated with sequential execution and the necessity of transforming raw data before it is stored or analyzed. The event-driven approach can contribute to increased latencies, particularly when handling high-frequency transactions, where data must be processed in rapid succession to maintain accuracy and operational effectiveness.

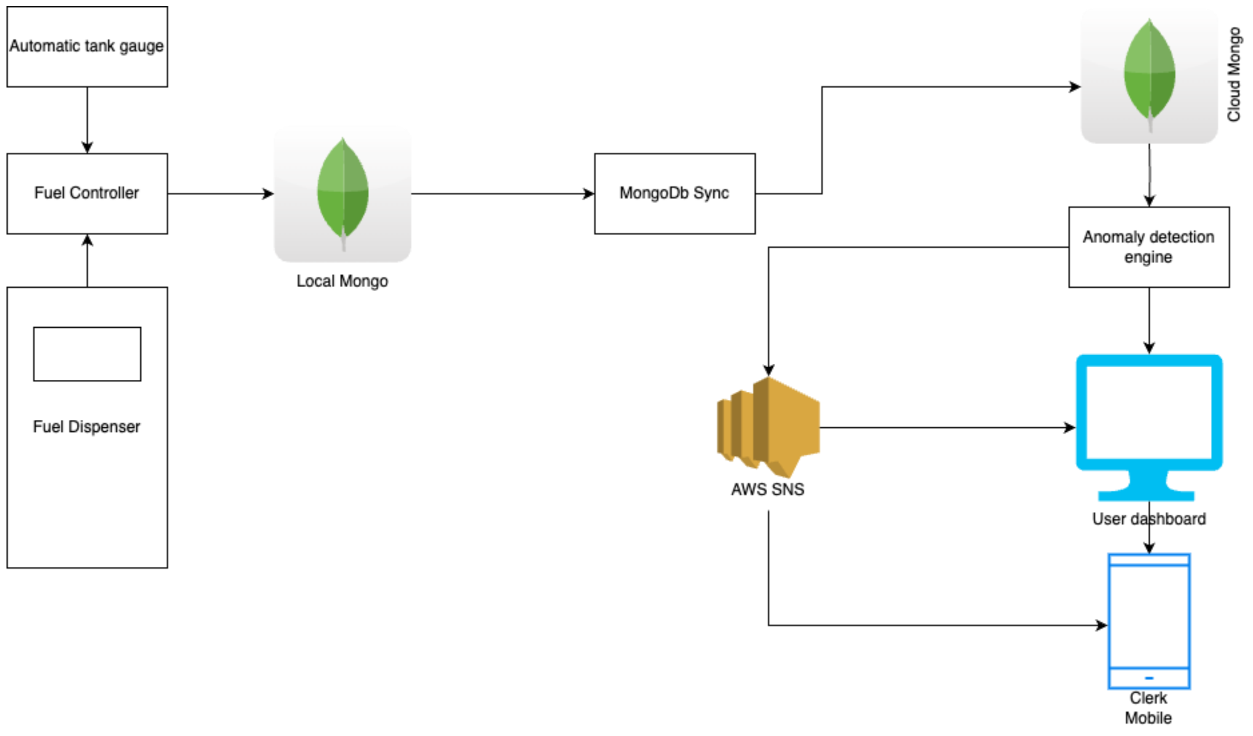
Furthermore, managing multiple event triggers and distributed processing workflows heightens the risk of inconsistencies, as data must be processed in the correct sequence to maintain coherence across different storage locations. The demand for real-time consistency across distributed environments presents an ongoing challenge, necessitating the development of robust synchronization techniques to prevent discrepancies between local and cloud databases. Fuel stations require immediate data availability and seamless integration across multiple locations, further reinforcing the need for a more direct and efficient solution

Database replication mechanisms have emerged as a viable alternative, offering lower latency and direct data consistency without the need for external event-driven processing layers. By leveraging advanced synchronization capabilities, database replication allows real-time data updates to be reflected instantly across geographically dispersed systems. This approach eliminates the dependency on event-driven processing, reducing reliance on additional cloud infrastructure and streamlining data flows. Unlike traditional streaming solutions, database-native replication operates on a continuous synchronization model, eliminating the need for intermediate event handling and batch processing. This shift results in reduced processing overhead and improved overall system responsiveness, making it a compelling alternative for fuel management systems.

The transition toward database-native replication mechanisms presents a promising direction for fuel management, particularly in environments where immediate transaction visibility and minimal processing latency are critical. Implementing direct database synchronization solutions enhances operational efficiency by eliminating redundant processing steps, reducing infrastructure costs, and ensuring real-time updates of fuel transactions, ATG readings, and dispenser activity logs. By enabling a real-time monitoring environment, this approach fosters more effective fuel station operations and ensures that all critical data remains up to date with minimal latency.

## 3. System Architecture

- **Local MongoDB Instance**: Resides within the fuel controller, capturing real-time transaction data.
- **MongoDB Sync Mechanism**: Configured to automatically synchronize changes with the cloud.
- **Cloud MongoDB Cluster**: Centralized storage for all fuel station data, enabling cross-site analytics.
- **Monitoring & Logging Module**: Tracks synchronization events and detects anomalies.



## 4. Implementation Strategy

The proposed implementation involves setting up a MongoDB replica set with local instances on fuel controllers and a primary cloud database. MongoDB's built-in synchronization feature allows bidirectional data flow, ensuring local transactions are updated in the cloud with minimal latency. Key configurations include:

- Deploying a local MongoDB instance with journaling enabled for durability.
- Using MongoDB Sync to establish a secure connection to the cloud.
- Implementing a conflict resolution policy to handle network failures.
- Optimizing indexes to reduce query execution times.

## 5. Case Study & Performance Evaluation

A pilot implementation was conducted at a mid-sized fuel station to analyze the real-world effectiveness of MongoDB Sync as a replacement for Kinesis and Lambda. The transition involved deploying a local MongoDB instance within the fuel controller, which was set up to synchronize with a cloud-based MongoDB cluster. This setup eliminated the need for event-driven ingestion and processing, thereby simplifying the overall architecture. The evaluation focused on key performance indicators, including data synchronization latency, consistency, and system resource consumption. Over a three-month period, transaction logs, ATG readings, and dispenser activities were continuously monitored to assess real-time data update accuracy and operational efficiency. The results demonstrated a significant reduction in latency, with MongoDB Sync achieving near-instantaneous updates compared to the previous architecture, which exhibited delays due to multiple processing stages. Additionally, resource consumption on both the local and cloud environments was optimized, leading to a more efficient and cost-effective fuel station data management solution. The study also identified best practices for deployment, including optimal index configurations and network stability measures to ensure uninterrupted synchronization. These findings establish MongoDB Sync as a robust and scalable alternative to traditional event-driven architectures for fuel station data management.

## 6. Results and Discussion

### 6.1 Pilot Implementation

Initial tests demonstrated that MongoDB Sync successfully updated cloud databases within 50ms of a local transaction, significantly faster than the previous Kinesis-Lambda pipeline, which averaged 300ms. The simplified architecture also reduced the number of required cloud services, lowering operational costs by 27%.

### 6.2 Performance Metrics

- Deploying a local MongoDB instance with journaling enabled for durability.
- Using MongoDB Sync to establish a secure connection to the cloud.
- Implementing a conflict resolution policy to handle network failures.
- Optimizing indexes to reduce query execution times.

## 7. Conclusion and Future Work

This study demonstrated that MongoDB Sync offers a viable alternative to Kinesis for real-time fuel transaction updates. By eliminating event-driven processing, fuel stations benefit from reduced latency and operational complexity. Future work includes expanding the study to larger deployments, implementing failover mechanisms, and exploring hybrid solutions that integrate MongoDB Sync with edge computing for enhanced resilience.

## 8. References

1. Krishnan, Hema & Elayidom, M.Sudheep & Santhanakrishnan, T.. (2016). MongoDB – a comparison with NoSQL databases. International Journal of Scientific and Engineering Research. 7. 1035-1037.
2. Tammaa, Ahmed. (2022). MongoDB Case Study on Forbes. 10.13140/RG.2.2.32766.46408.
3. Mungekar, Akshay. (2019). Data Storage and Management Project. 10.13140/RG.2.2.27354.18880.
4. Győrödi, Cornelia &Gyorodi, Robert &Pecherle, George & Olah, Andrada. (2015). A Comparative Study: MongoDB vs. MySQL. 10.13140/RG.2.1.1226.7685.
5. Gorasiya, Darshankumar. (2019). Quantitative Performance Evaluation of Cloud-Based MySQL (Relational) Vs. MongoDB (NoSQL) Database with YCSB. 10.13140/RG.2.2.29628.59528.
6. Heydari Beni, Emad. (2015). Finding efficient Shard Keys with a learning process on query logs in Database Sharding. 10.13140/RG.2.1.1512.0486.
7. Sarkar, Anindita& Sanyal, Madhupa& Chattopadhyay, Samiran& Mondal, Dr-Kartick. (2017). Comparative Analysis of Structured and Un-Structured Databases. 226-241. 10.1007/978-981-10-6430-2_18.
8. Prasad, Aashish. (2018). HBase vs Mongo DB. 10.13140/RG.2.2.15766.80968.
9. Pandey, Rachit. (2020). Performance Benchmarking and Comparison of Cloud-Based Databases MongoDB (NoSQL) Vs MySQL (Relational) using YCSB. 10.13140/RG.2.2.10789.32484.